

MODULE – 3

Naive Bayes, Text Classification and Sentiment

Naive Bayes, Text Classification and Sentiment: Naive Bayes Classifiers, Training the Naive Bayes Classifier, Worked Example, Optimizing for Sentiment Analysis, Naive Bayes for Other Text Classification Tasks, Naive Bayes as a Language Model.

Textbook 2: Ch. 4.

Introduction

- Classification, heart of both human and machine intelligence ⑦ assigning a category to an input
- Deciding what letter, word, or image has been presented to our senses, recognizing faces or voices, sorting mail, assigning grades to homeworks;
- **Naïve Bayes algorithm** for *text categorization*: the task of *assigning a label or category to an entire text or document*.
- Common text categorization tasks:
 1. **Sentiment analysis**, the extraction of sentiment, the *positive or negative* orientation that a writer expresses toward some object.
 - A review of a movie, book, or product on the web.

Example: + ... *any characters and richly applied satire, and some great plot twists*

- *It was pathetic. The worst part about it was the boxing scenes ...*
- + ... *awesome caramel sauce and sweet toasty almonds. I love this place!*
- ... *awful pizza and ridiculously overpriced ...*

Words like great, richly, awesome, and pathetic, and awful and ridiculously are very informative cues: unification is based on the functional specifications of the verb, which predicts the overall sentence structure.

2. Spam detection:

- Binary classification task of assigning an email to one of the two classes spam or not-spam.
- Many lexical and other features can be used to perform classification.

Example: Suspicious of an email containing phrases like “online pharmaceutical” or “WITHOUT ANY COST” or “Dear Winner”.

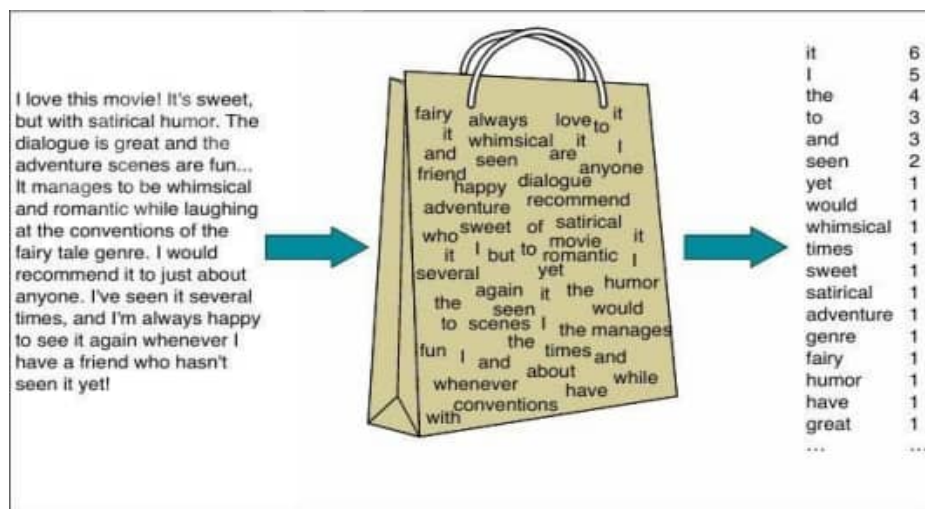
3. **Assigning a library subject category or topic label to a text:** Various sets of subject categories exist. Deciding whether a research paper concerns epidemiology, embryology, etc..is an important component of information retrieval.

Supervised Learning:

- The most common way of doing text classification in language processing is supervised learning.
- In supervised learning, we have a data set of input observations, each associated with some correct output (a ‘supervision signal’).
- The goal of the algorithm is to learn how to map from a new observation to a correct output.
- We have a training set of N documents that have each been hand labeled with a class: $\{(d_1, c_1) \dots (d_N, c_N)\}$. Our goal is to learn a classifier that is capable of mapping from a new document d to its correct class $c \in C$, where C is some set of useful document classes.

3.1 Naive Bayes Classifiers

The intuition of the classifier is shown in Fig. 1. We represent a text document as if it were a bag of words, that is, an unordered set of words with their position ignored, keeping only their frequency in the document.



Instead of representing the word order in all the phrases like “I love this movie” and “I would recommend it”, we simply note that the word I occurred 5 times in the entire excerpt, the word it 6 times, the words love, recommend, and movie once, and so on.

- Naive Bayes is a probabilistic classifier.

- For a document d , out of all classes $c \in C$ the classifier returns the class $\hat{C}$ which has the maximum posterior probability given the document.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) \quad (1)$$

Use Bayes' rule to break down any conditional probability $P(x|y)$ into three other probabilities:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \quad (2)$$

We can then substitute Eq.2 into Eq.1 to get Eq.3

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)} \quad (3)$$

Since $P(d)$ doesn't change for each class, we can conveniently simplify Eq. 3 by dropping the denominator.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c) \quad (4)$$

We call Naive Bayes a **generative** model, Eq. 4 can be read as class is sampled from $P(c)$, and then the words are generated by sampling from $P(d|c)$ and a document is generated.

Eq. 4 states, we compute the most probable class $\hat{C}$ given some document d by choosing the class which has the highest product of two probabilities: the **prior probability** of the class $P(c)$ and the **likelihood** of the document $P(d|c)$:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \underbrace{P(d|c)}_{\text{likelihood}} \underbrace{P(c)}_{\text{prior}} \quad (5)$$

we can represent a document d as a set of features $f_1, f_2, \dots, f_n$:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \underbrace{P(f_1, f_2, \dots, f_n|c)}_{\text{likelihood}} \underbrace{P(c)}_{\text{prior}} \quad (6)$$

Eq. 6 is still too hard to compute directly: without some simplifying assumptions, estimating the probability of every possible combination of features (for example, every possible set of words and positions) would require huge numbers of parameters and impossibly large training sets.

Naive Bayes classifiers therefore make two simplifying assumptions.

The first is the bag-of-words assumption, that the features $f_1, f_2, \dots, f_n$ only encode word identity and not position.

The second is commonly called the naive Bayes assumption, the conditional independence assumption that the probabilities $P(f_i|c)$ are independent given the class c .

Therefore,
$$P(f_1, f_2, \dots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \dots \cdot P(f_n|c) \quad (7)$$

The final equation for the class chosen by a naive Bayes classifier is:

$$c_{NB} = \operatorname{argmax}_{c \in C} P(c) \prod_{f \in F} P(f|c) \quad (8)$$

To apply the naive Bayes classifier to text, we will use *each word in the documents as a feature*, as suggested above, and we consider each of the words in the document by walking an *index* through every word position in the document:

$$c_{NB} = \operatorname{argmax}_{c \in C} P(c) \prod_{i \in \text{positions}} P(w_i|c) \quad (9)$$

Naive Bayes calculations, like calculations for language modelling, are done in log space, to avoid underflow and increase speed. Thus Eq. 9 is generally instead expressed as,

$$c_{NB} = \operatorname{argmax}_{c \in C} \log P(c) + \sum_{i \in \text{positions}} \log P(w_i|c) \quad (10)$$

Eq. 10 computes the predicted class as a linear function of input features. Classifiers that use a linear combination of the inputs to make a classification decision -like naive Bayes and also logistic regression are called linear classifiers.

3.2 Training the Naive Bayes Classifier

How can we learn the probabilities $P(c)$ and $P(f_i|c)$?

To learn class priori $P(c)$: What percentage of the documents in our training set are in each class c .

Let N_c be the number of documents in our training data with class c . N_{doc} be the total number of documents. Then,

$$\hat{P}(c) = \frac{N_c}{N_{doc}} \quad (11)$$

To learn the probability $P(f_i|c)$:

We'll assume a feature is just the existence of a word in the document's bag of words, and so we'll want $P(w_i|c)$, we compute as the fraction of times the word w_i appears among all words in all documents of topic c .

Concatenate all documents with category c into one big "category c " text. Then we use the frequency of w_i in this concatenated document to give a maximum likelihood estimate of the probability:

i.e (12)

Here the $\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)}$ vocabulary V
total number of unique words in your vocabulary in all classes, not just the words in one class c .

Issues with training:

1. Zero Probability problem with maximum likelihood training:

Imagine we are trying to estimate the likelihood of the word "fantastic" given class *positive*, but suppose there are *no training documents that both contain the word "fantastic" and are classified as positive*. Perhaps the word "fantastic" happens to occur (sarcastically?) in the class *negative*. In such a case the probability for this feature will be zero:

$$\hat{P}(\text{"fantastic"}|\text{positive}) = \frac{\text{count}(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0 \quad (13)$$

Since naive Bayes naively multiplies all the feature likelihoods together, zero probabilities in the likelihood term for any class will cause the probability of the class to be zero, no matter the other evidence!

To solve this, we use something called **Laplace smoothing** (or add-one smoothing). Instead of:

$$P(w_i | c) = \frac{\text{count}(w_i, c)}{\text{total words in } c}$$

We do:

$$P(w_i | c) = \frac{\text{count}(w_i, c) + 1}{\text{total words in } c + V} \quad (14)$$

Where:

- $+1$ ensures no word ever gets zero probability.
- V is the total number of unique words in your vocabulary.

Now "fantastic" will still get a **very small** probability in the "positive" class — but **not zero**.

2. Words that occur in our test data but are not in our vocabulary:

- Remove them from the test document and not include any probability for them at all. Some systems choose to completely ignore another class of words: stop words, very frequent words like *the* and *a*.

- Defining the top 10-100 vocabulary entries as stop words, or alternatively by using one of the many predefined stop word lists available online. Then each instance of these stop words is simply removed from both training and test documents.
- However, using a stop word list doesn't improve performance, and so it is more common to make use of the entire vocabulary.

```

function TRAIN NAIVE BAYES(D, C) returns V, log P(c), log P(w|c)

for each class c ∈ C           # Calculate P(c) terms
    Ndoc = number of documents in D
    Nc = number of documents from D in class c
    logprior[c] ← log  $\frac{N_c}{N_{doc}}$ 
    V ← vocabulary of D
    bigdoc[c] ← append(d) for d ∈ D with class c
    for each word w in V           # Calculate P(w|c) terms
        count(w,c) ← # of occurrences of w in bigdoc[c]
        loglikelihood[w,c] ← log  $\frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$ 
return logprior, loglikelihood, V

function TEST NAIVE BAYES(testdoc, logprior, loglikelihood, C, V) returns best c

for each class c ∈ C
    sum[c] ← logprior[c]
    for each position i in testdoc
        word ← testdoc[i]
        if word ∈ V
            sum[c] ← sum[c] + loglikelihood[word,c]
return argmaxc sum[c]

```

Fig. The naïve Bayes algorithm, using add-1 smoothing. To use add-smoothing instead, change the +1 to + α for log likelihood counts in training.

3.3 Worked example:

Let's use a sentiment analysis domain with the two classes *positive* (+) and *negative* (-), and take the following miniature training and test documents simplified from actual movie reviews.

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

Step 1: Prior $P(c)$ for the two classes is computed as per equation $\hat{P}(c) = \frac{N_c}{N_{doc}}$

$$P(-) = 3/5 \quad P(+) = 2/5$$

Step 2: The word “**with**” doesn't occur in the training set, so we drop it completely.

Step 3: The likelihoods from the training set for the remaining three words "predictable", "no", and "fun", are as follows:

$$P(w_i | c) = \frac{\text{count}(w_i, c) + 1}{\text{total words in } c + V}$$

$$\begin{aligned} P(\text{"predictable"}|-) &= \frac{1+1}{14+20} & P(\text{"predictable"}|+) &= \frac{0+1}{9+20} \\ P(\text{"no"}|-) &= \frac{1+1}{14+20} & P(\text{"no"}|+) &= \frac{0+1}{9+20} \\ P(\text{"fun"}|-) &= \frac{0+1}{14+20} & P(\text{"fun"}|+) &= \frac{1+1}{9+20} \end{aligned}$$

Step 4: For the test sentence $S = \text{"predictable with no fun"}$, after removing the word 'with', the

chosen class, via Eq. 9: $c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i \in \text{positions}} P(w_i | c)$ is therefore computed as follows:

$$\begin{aligned} P(-)P(S|-) &= \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5} \\ P(+)P(S|+) &= \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5} \end{aligned}$$

3.4 Optimizing for Sentiment Analysis

While standard *naive Bayes* text classification can *work well for sentiment analysis*, some *small changes* are generally employed that *improve performance*.

3.4.1 Clip the word counts (duplicate words) in each document at 1:

- Remove all duplicate words before concatenating them into the single big document during training and we also remove duplicate words from test documents.
- This variant is called *binary multinomial naive Bayes* or *binary naive Bayes*.
- Example:

		NB Counts		Binary Counts		
		+	-	+	-	
Four original documents:						
-	it was pathetic the worst part was the boxing scenes	and	2	0	1	0
		boxing	0	1	0	1
-	no plot twists or great scenes	film	1	0	1	0
+	and satire and great plot twists	great	3	1	2	1
+	great scenes great film	it	0	1	0	1
		no	0	1	0	1
		or	0	1	0	1
		part	0	1	0	1
		pathetic	0	1	0	1
		plot	1	1	1	1
		satire	1	0	1	0
		scenes	1	2	1	2
		the	0	2	0	1
		twists	1	1	1	1
		was	0	2	0	1
		worst	0	1	0	1
After per-document binarization:						
-	it was pathetic the worst part boxing scenes					
-	no plot twists or great scenes					
+	and satire great plot twists					
+	great scenes film					

Fig. An example of binarization for the binary naive Bayes algorithm

3.4.2 Deal with negation.

Consider the difference between *I really like this movie* (positive) and *I didn't like this movie* (negative). Similarly, negation can modify a negative word to produce a positive review (*don't dismiss this film, doesn't let us get bored*).

Solution: Prepend the prefix **NOT** to every word after a token of logical negation (n't, not, no, never) until the next punctuation mark.

Thus the phrase: **didn't like this movie , but I**

becomes: **didnt NOT_like NOT_this NOT_movie , but I**

'words' like *NOT_like*, *NOT_recommend* will thus occur more often in negative document and act as cues for negative sentiment, while words like *NOT_bored*, *NOT_dismiss* will acquire positive associations.

3.4.3 Insufficient labelled training data:

Derive the positive and negative word features from **sentiment lexicons (corpus)**, **lists of words that are pre-annotated** with positive or negative sentiment.

For example, the MPQA lexicon corpus subjectivity lexicon has 6885 words each marked for whether it is strongly or weakly biased positive or negative. Some examples:

+ : *admirable, beautiful, confident, dazzling, ecstatic, favor, glee, great*

- : *awful, bad, bias, catastrophe, cheat, deny, envious, foul, harsh, hate*

3.5 Naive Bayes for other text classification tasks

3.5.1 Spam Detection and Naïve Bayes

Spam detection—deciding whether an email is unsolicited bulk mail—was one of the earliest applications of naïve Bayes in text classification (Sahami et al., 1998). Rather than treating all words as individual features, effective systems often use predefined sets of words or patterns, along with non-linguistic features.

For instance, the open-source tool SpamAssassin uses a range of handcrafted features:

- Specific phrases like "one hundred percent guaranteed"
- Regex patterns like mentions of millions of dollars
- Structural properties like HTML with a low text-to-image ratio
- Non-linguistic metadata, such as the email's delivery path

Other examples of SpamAssassin features include:

- Subject lines written entirely in capital letters
- Urgent phrases like "urgent reply"
- Keywords such as "online pharmaceutical"
- HTML anomalies like unbalanced head tags
- Claims such as "you can be removed from the list"

3.5.2 Language Identification

In contrast, tasks like language identification rely less on words and more on subword units like character n-grams or even byte n-grams. These can capture statistical patterns at the start or end of words, especially when spaces are included as characters.

A well-known system, `langid.py` (Lui & Baldwin, 2012), starts with all possible n-grams of lengths 1–4 and uses feature selection to narrow down to the 7,000 most informative.

Training data for language ID systems often comes from multilingual sources such as Wikipedia (in 68+ languages), newswire, and social media. To capture regional and dialectal diversity, additional corpora include:

- Geo-tagged tweets from Anglophone regions like Nigeria or India
- Translations of the Bible and Quran
- Slang from Urban Dictionary
- Corpora of African American Vernacular English (Blodgett et al., 2016)

These diverse sources help models capture the full range of language use across different communities and contexts (Jurgens et al., 2017).

3.6 Naive Bayes as a Language Model

- Naive Bayes classifiers can use any sort of feature: dictionaries, URLs, email addresses, network features, phrases, and so on.
- A naive Bayes model can be viewed as a set of class-specific unigram language models, in which the model for each class instantiates a unigram language model.
- Assign a probability to each word $P(\text{word}|c)$, the model also assigns a probability to each

$$P(s|c) = \prod_{i \in \text{positions}} P(w_i|c) \quad (15)$$

sentence:

Example: Consider a naive Bayes model with the classes positive (+) and negative (-) and the following model parameters:

w	P(w +)	P(w -)
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...	...	...

Each of the two columns above instantiates a language model that can assign a probability to the sentence ***“I love this fun film”***:

$$P(\text{“I love this fun film”}+) = 0.1 * 0.1 * 0.01 * 0.05 * 0.1 = 5 * 10^{-7}$$

$$P(\text{“I love this fun film”}-) = 0.2 * 0.001 * 0.01 * 0.005 * 0.1 = 1.0 * 10^{-9}$$

The positive model assigns a higher probability to the sentence: $P(s|\text{pos}) > P(s|\text{neg})$.

Note: This is just the likelihood part of the naive Bayes model; once we *multiply in the prior* a full naive Bayes model might well make a different classification decision.

3.7 Evaluation: Precision, Recall, F-measure

Text classification evaluation often starts with binary detection tasks.

Example 1: Spam Detection

- Goal: Label each text as spam (positive) or not spam (negative).
- Need to compare:
 - System's prediction
 - Gold label (human-defined correct label)

Example 2: Social Media Monitoring for a Brand

- Scenario: CEO of Delicious Pie Company wants to track mentions on social media.
- Build a system to detect tweets about Delicious Pie.
- Positive class: Tweets about the company.
- Negative class: All other tweets.

Why we need metrics:

- To evaluate how well a system (e.g., spam detector or pie-tweet detector) performs.
- **Confusion Matrix:**

		gold standard labels		
		gold positive	gold negative	
system output labels	system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
	system negative	false negative	true negative	
		recall = $\frac{tp}{tp+fn}$		accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

- A table that compares **system predictions** vs. **gold (human) labels**.

- Each cell represents a type of outcome:

□ **True Positive (TP):** Correctly predicted positives (e.g., actual spam labeled as spam).

□ **False Negative (FN):** Actual positives incorrectly labeled as negative (e.g., spam labeled as non-spam).

- **Accuracy:**

- Formula: **(Correct predictions) / (Total predictions)**.
- Appears useful but **misleading for unbalanced classes**.

- **Why accuracy can fail:**

- Real-world data is often skewed (e.g., most tweets are not about pie).
- Example:

□ 1,000,000 tweets → only 100 about pie.

□ A naive classifier labels all tweets as "not about pie".

□ Result: 99.99% accuracy, but **0 useful results**.

- **Conclusion:** Accuracy is not a reliable metric when the positive class is rare. That's why, instead of relying on **accuracy**, we often use two more informative metrics: **precision** and **recall** (as shown in Fig).

- **Precision** measures the percentage of items labeled as positive by the system that are actually positive (according to human-annotated “gold” labels).

$$\text{Precision} = \text{true positives} / (\text{true positives} + \text{false positives})$$

- **Recall** measures the percentage of actual positive items that were correctly identified by the system.

$$\text{Recall} = \text{true positives} / (\text{true positives} + \text{false negatives})$$

These metrics address the issue with the “**nothing is pie**” classifier. Despite its seemingly excellent **99.99% accuracy**, it has a **recall of 0** —because it misses all 100 true positive cases, identifying none. Its **precision** is also meaningless, since it detects nothing (since there are no true positives, and 100 false negatives, the recall is 0/100).

Unlike accuracy, **precision and recall focus on true positives**, helping us measure how well the system finds the things it's actually supposed to detect.

To combine both precision and recall into a single metric, we use the **F-measure** (van Rijsbergen, 1975), with the most common version being the **F1 score**:

$$F_{\beta} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

The β parameter differentially weights the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is the most frequently used metric, and is called $F_{\beta=1}$ or just F_1 :

$$F_1 = \frac{2PR}{P+R} \quad (16)$$

3.7.1 Evaluating with more than two classes

For sentiment analysis we generally have 3 classes (positive, negative, neutral) and even more classes are common for tasks like part-of-speech tagging, word sense disambiguation, semantic role labeling, emotion detection, and so on. Luckily the naive Bayes algorithm is already a multi-class classification algorithm.

Consider the sample confusion matrix for a hypothetical 3-way one-of email categorization decision (urgent, normal, spam) shown in Fig. The matrix shows, for example, that the system mistakenly labeled one spam document as urgent, and we have shown how to compute a distinct precision and recall value for each class.

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2 .

In order to derive a single metric that tells us how well the system is doing, we can combine these values in two ways.

1. In **macroaveraging**, we compute the performance for each class, and then average over classes.
2. In **microaveraging**, we collect the decisions for all classes into a single confusion matrix, and then compute precision and recall from that table.

Fig. shows the **confusion matrix for each class separately**, and shows the computation of *microaveraged* and *macroaveraged* precision.

Class 1: Urgent			Class 2: Normal			Class 3: Spam			Pooled		
	true urgent	true not		true normal	true not		true spam	true not		true yes	true no
system urgent	8	11	system normal	60	55	system spam	200	33	system yes	268	99
system not	8	340	system not	40	212	system not	51	83	system no	99	635
precision = $\frac{8}{8+11} = .42$			precision = $\frac{60}{60+55} = .52$			precision = $\frac{200}{200+33} = .86$			microaverage precision = $\frac{268}{268+99} = .73$		
macroaverage precision = $\frac{.42+.52+.86}{3} = .60$											

As the figure shows, a microaverage is dominated by the more frequent class (in this case spam), since the counts are pooled. The macroaverage better reflects the statistics of the smaller classes, and so is more appropriate when performance on all the classes is equally important

3.8 Test sets and Cross-validation

Training & Testing for Text Classification:

1. Standard Procedure:

- Train the model on the **training set**. ○ Use the **development set (devset)** to tune parameters and choose the best model.
- Evaluate the final model on a **separate test set**.

2. Issue with Fixed Splits:

- Fixed training/dev/test sets may lead to small dev/test sets.
- Smaller test sets might not be **representative** of overall performance.

3. Solution – Cross-Validation (as shown in Fig):

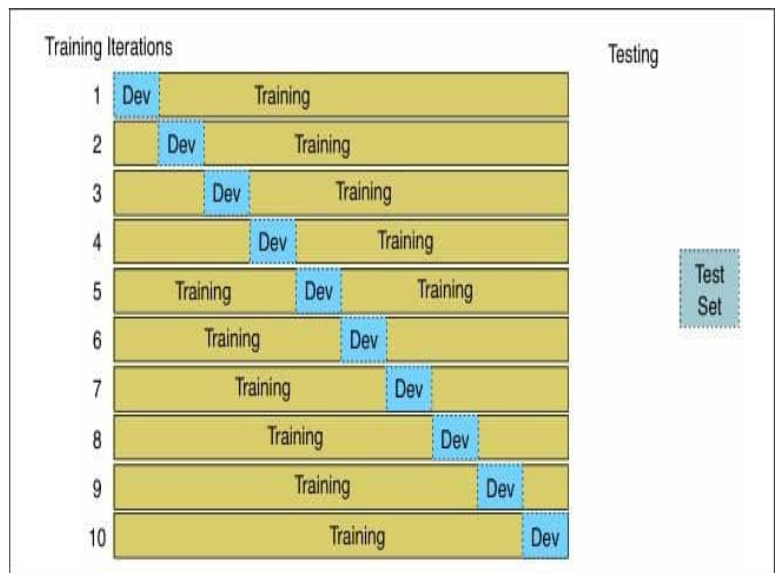
- **Cross-validation** allows use of all data for training and testing.
- Process:
 - Split data into **k folds**.
 - For each fold:
 - Train on **k-1 folds**, test on the **remaining fold**.
 - Repeat **k times**, average the test errors.
- Example: **10-fold cross-validation** (train on 90%, test on 10%, repeated 10 times).

4. Limitation of Cross-Validation:

- All data is used for testing → can't analyze the data in advance (avoiding "peeking").
- Looking at data is important for feature design in **NLP systems**.

5. Common Compromise:

- Split off a **fixed test set**.
- Do **10-fold cross-validation on the training set**.
- Use test set only for **final evaluation**.



3.9 Statistical Significance Testing

- When building NLP systems, we often need to compare performance between two systems (e.g., a new model vs. an existing one).
- Simply observing different scores (e.g., accuracy, F1) isn't enough — we need to know if the difference is statistically significant.
- This is where statistical hypothesis testing comes in.
- Inspired by Dror et al. (2020) and Berg-Kirkpatrick et al. (2012), these tests help determine if the observed improvement is real or due to chance.
- Example:
 - Classifier A (e.g., logistic regression) vs. Classifier B (e.g., naive Bayes).
 - Metric M (e.g., F1-score), tested on dataset x.
 - Let $M(A, x)$ be the score for A, and $\delta(x)$ be the difference in performance between A and B.

$$\delta(x) = M(A, x) - M(B, x) \quad (19)$$

Understanding Effect Size and Significance

- We want to know if $\delta(x) > 0$, meaning A (logistic regression) performs better than B (naive Bayes).
- $\delta(x)$ is the **effect size** — larger δ means a bigger performance gap.
- But a positive δ alone isn't enough.
 - Example: A has 0.04 higher F1 than B — is that meaningful?
- **Problem:** The difference might be due to chance on this specific test set.
- What we really want to know:

- Would A still outperform B on **another test set** or under **different conditions**?
- That's why we need **statistical testing**, not just raw differences.

Statistical Hypothesis Testing Paradigm

- We compare models by setting up two formal hypotheses:

$$\begin{aligned} H_0 &: \delta(x) \leq 0 \\ H_1 &: \delta(x) > 0 \end{aligned} \tag{20}$$

- **Null hypothesis (H_0):** There's **no real difference** between A and B — any observed difference is due to **chance**.
- **Alternative hypothesis (H_1):** There **is** a real performance difference between A and B.
- Statistical tests help us decide whether to **reject H_0** in favor of H_1 based on the data.

Null Hypothesis and p-value

- **Null hypothesis (H_0):** Assumes $\delta(x) \leq 0$ — A is **not better** than B.
- We want to see if we can **reject H_0** and support **H_1** (that A *is* better).
- We imagine $\delta(x)$ over **many possible test sets**.
- The **p-value** measures how likely we are to observe our $\delta(x)$, or a larger one, **if H_0 were true**. $P(\delta(X) \geq \delta(x) | H_0 \text{ is true})$ (21)
- A **low p-value** suggests our result is **unlikely due to chance**, supporting H_1 .

Interpreting p-values and Statistical Testing in NLP

- The **p-value** is the probability of observing a performance difference $\delta(x)$ (or larger), **assuming A is not better than B** (null hypothesis H_0).
- If $\delta(x)$ is large (e.g., A's F1 = 0.9 vs. B's = 0.2), it's **unlikely** under $H_0 \rightarrow$ **low p-value** $\rightarrow$ we **reject H_0** .
- If $\delta(x)$ is small, it's more **plausible under H_0** $\rightarrow$ **higher p-value** $\rightarrow$ we may **fail to reject H_0** .

What Counts as “Small”? ○ Common p-value thresholds: **0.05** or **0.01**

- If **$p < \text{threshold}$** , the result is considered **statistically significant** (we reject H_0 and conclude A is likely better than B).

How Do We Compute the p-value in NLP?

- NLP avoids **parametric tests** (like t-tests or ANOVAs) because they assume certain distributions that often don't apply.
- Instead, we use **non-parametric tests** that rely on **sampling methods**.

Key Idea:

- Simulate many variations of the experiment (e.g., using different test sets x').
- Compute $\delta(x')$ for each $\rightarrow$ this gives a **distribution** of δ values.
- If the **observed** $\delta(x)$ is in the top 1% (i.e., **p-value** < 0.01), it's unlikely under $H_0 \rightarrow$ **reject H_0** .

Common Non-Parametric Tests in NLP:

1. **Approximate Randomization** (Noreen, 1989)
2. **Bootstrap Test** (paired version is most common)
 - Compares **aligned outputs** from two systems (e.g., A vs. B on the **same inputs** x_i).
 - Measures how consistently one system outperforms the other across samples.

3.9.1 The Paired Bootstrap Test

The bootstrap test is a flexible, non-parametric method that can be applied to any evaluation metric—like precision, recall, F1, or BLEU.

What is bootstrapping?

It involves repeatedly sampling with replacement from an original dataset to create many "bootstrap samples" or virtual test sets. The key assumption is that the original sample is representative of the larger population.

Example

Imagine a small classification task with 10 test documents. Two classifiers, A and B, are evaluated:

- Each document outcome falls into one of four categories:
 - Both A and B correct Both incorrect
 - A correct, B wrong
 - A wrong, B correct
- If A has 70% accuracy and B has 50%, then the performance difference $\delta(x) = 0.20$.

How bootstrap works:

1. Generate a large number (e.g., 100,000) of new test sets by sampling 10 documents *with replacement* from the original set.
2. For each virtual test set, recalculate the accuracy difference between A and B.
3. Use the distribution of these differences to estimate a **p-value**, telling us how likely the observed $\delta(x)$ is under the null hypothesis (that A is not better than B).

This helps determine whether the observed performance difference is statistically significant or just due to random chance.

	1	2	3	4	5	6	7	8	9	10	A%	B%	$\delta()$
x	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.70	.50	.20
$x^{(1)}$	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.60	.60	.00
$x^{(2)}$	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.60	.70	-.10
...													
$x^{(b)}$													

Figure: The paired bootstrap test: Examples of b pseudo test sets $x^{(i)}$ being created from an initial true test set x . Each pseudo test set is created by sampling $n = 10$ times with replacement; thus an individual sample is a single cell, a document with its gold label and the correct or incorrect performance of classifiers A and B.

With the **b bootstrap test sets**, we now have a sampling distribution to analyze whether A's advantage is due to chance. Following **Berg-Kirkpatrick et al. (2012)**, we assume the **null hypothesis (H_0)**—that A is *not* better than B—so the average $\delta(x)$ should be zero or negative. If our observed $\delta(x)$ is much higher, it would be surprising under H_0 . To measure this, we calculate the **p-value** by checking how often the sampled $\delta(x_i)$ values exceed the observed $\delta(x)$.

$$\text{p-value}(x) = \frac{1}{b} \sum_{i=1}^b \mathbf{1}(\delta(x^{(i)}) - \delta(x) \geq 0)$$

We use the notation $\mathbf{1}(x)$ to mean “1 if x is true, and 0 otherwise.” Although the expected value of $\delta(X)$ over many test sets is 0, this isn't true for bootstrapped test sets due to the bias in the original test set, so we compute the p-value by counting how often $\delta(x^{(i)})$ exceeds the expected $\delta(x)$ by $\delta(x)$ or more.

$$\begin{aligned} \text{p-value}(x) &= \frac{1}{b} \sum_{i=1}^b \mathbf{1}(\delta(x^{(i)}) - \delta(x) \geq \delta(x)) \\ &= \frac{1}{b} \sum_{i=1}^b \mathbf{1}(\delta(x^{(i)}) \geq 2\delta(x)) \end{aligned} \tag{22}$$

If we have 10,000 test sets and a threshold of 0.01, and in 47 test sets we find $\delta(x^{(i)}) \geq 2\delta(x)$, the p-value of 0.0047 is smaller than 0.01. This suggests the result is surprising, allowing us to reject the null hypothesis and conclude A is better than B.

```

function BOOTSTRAP(test set  $x$ , num of samples  $b$ ) returns  $p\text{-value}(x)$ 
  Calculate  $\delta(x)$  # how much better does algorithm A do than B on  $x$ 
   $s = 0$ 
  for  $i = 1$  to  $b$  do
    for  $j = 1$  to  $n$  do # Draw a bootstrap sample  $x^{(i)}$  of size  $n$ 
      Select a member of  $x$  at random and add it to  $x^{(i)}$ 
      Calculate  $\delta(x^{(i)})$  # how much better does algorithm A do than B on  $x^{(i)}$ 
       $s \leftarrow s + 1$  if  $\delta(x^{(i)}) \geq 2\delta(x)$ 
   $p\text{-value}(x) \approx \frac{s}{b}$  # on what % of the  $b$  samples did algorithm A beat expectations?
  return  $p\text{-value}(x)$  # if very few did, our observed  $\delta$  is probably not accidental

```

Fig. A version of the paired bootstrap algorithm

The full algorithm for the bootstrap is shown in Fig. It is given a test set x , a number of samples b , and counts the percentage of the b bootstrap test sets in which $\delta(x^{(i)}) > 2\delta(x)$. This percentage then acts as a one-sided empirical p-value.

3.10 Avoiding Harms in Classification (Summary)

- **Classifiers can cause harm**, including **representational harms** (e.g., reinforcing stereotypes).
 - Example: Sentiment analysis systems rated sentences with African American names more negatively than identical ones with European American names.
- **Toxicity classifiers** may falsely label non-toxic content as toxic, especially when it references marginalized groups or dialects (e.g., AAVE), leading to **silencing**.
- Harms can arise from:
 - **Biased training data** ◦ **Biased labels or resources** (e.g., lexicons, embeddings) ◦ **Model design choices**
- **No universal fix exists**, so transparency is key.
- A proposed solution: release **model cards** (Mitchell et al., 2019), which include:
 - Training algorithms and parameters
 - Training data sources, motivation, and preprocessing
 - Evaluation data sources, motivation, and preprocessing
 - Intended use and users
 - Model performance across different demographic or other groups and environmental situations